A função realloc(): realocando memória dinamicamente e a calloc()

Neste tutorial de nossa apostila de C, iremos aprender o que é a função realloc(), para que serve o realocamento de memória, vamos ver como fazer isso através de exemplos de códigos comentados, alar sobre a função calloc(), além de dar mais dicas sobre alocação dinâmica de memória.

Esta função completa nosso estudo sobre a alocação dinâmica, junto com os artigos sobre a função malloc() e sobre a liberação de memória com a função free().

Problemas com a malloc()

Agora que já mostramos que nem sempre a malloc() é a solução mais eficiente e produtiva, vamos mostrar como tais problemas podem ser contornados, através do uso da função realloc().

Como o próprio nome diz, ela realoca um espaço de memória.
Ou seja, para realocar é necessário que algo tenha sido alocado. Então, antes de ver a sintaxe da realloc() podemos concluir que para usar ela é necessário ter um ponteiro que foi usado para alocar um espaço de memória.
A realloc(), assim como a malloc(), retorna um endereço com um novo bloco de memória.

Seja 'ptr' esse ponteiro, a sintaxe para o uso da função realloc() é:

```
realloc(ptr, numero_bytes);
```

O 'numero_bytes' é o número de bytes que queremos realocar. É exatamente da mesma maneira que fizemos como na malloc(), onde geralmente fazemos 'n * sizeof(tipo)'.

Outra coisa que devemos lembrar é que um ponteiro deve receber o endereço da realloc(), ou seja, não devemos usar ela de maneira 'solta', alguém devem receber seu retorno. É um erro comum simplesmente escrever algo do tipo:

realloc(ponteiro, n*sizeof(int))

E achar que agora o ponteiro 'ponteiro' foi usado para alocar aquele espaço de memória, quando não foi.
Se quisermos fazer isso, devemos capturar o retorno:
ponteiro =(int *) realloc(ponteiro, n*sizeof(int));

Note que devemos usar o cast de ponteiros aqui também, como na malloc.
Ou seja, se o ponteiro 'ptr' aponta para float, fazemos:
ptr = (float *) realloc(ptr, n*sizeof(float));
Se for do tipo char:
ptr = (char *) realloc(ptr, n*sizeof(char));

Memória não alocada

Antes de mostrar um exemplo prático do uso da função realloc(), vamos fazer uma pausa neste tutorial de nossa apostila para dar mais uma dica, um bom hábito de programação que você deve ter.

Até o momento estávamos agindo como se a memória fosse sempre alocada, o que geralmente ocorre, pois as máquinas atuais possuem muito espaço em memória, e para as aplicações simples e básicas de nosso curso, precisamos de alocar pouca memória.

Porém quando você se tornar profissional e for criar aplicativos mais complexos e robustos, ou for trabalhar com dispositivos com pouca memória (como microcontroladores), verá que nem sempre existe espaço suficiente de memória.

Quando pedimos memória e não há espaço suficiente, a função retorna o endereço NULL.
Por isso, faça sempre um teste condicional IF após alocação de memória, para tratar o caso em que não exista espaço suficiente de memória.

Como exemplo, vamos tentar alocar um espaço de memória absurdamente grande, e vemos a mensagem de erro:

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
 int* ptr = (int *) malloc(1000000000000000000);

 if(ptr == NULL)
```

Artigos populares

Gerando números aleatórios em C: rand, srand e seed
Você pode nunca ter ficado atento para isso, mas números aleatórios são vitais em praticamente todos os ramos da computação; Em jogos que...

Os tipos float e double - números decimais (ou reais) em C
No artigo passado de nosso curso de C, estudamos sobre o tipo inteiro (int) , como declarar, imprimir e inicializar tal tipo de dado. Agor...

O que são vetores, como declarar e quando usar
Dando início a mais uma importante unidade em nosso curso online e gratuito de C, vamos iniciar nossos estudos sobre as estruturas de dados....


Criando e compilando seu primeiro programa na Linguagem C
No artigo passado do curso C Progressivo você baixou e instalou a IDE Clode::Blocks, que é o programa necessário (mais recomendado e melhor...

Operações matemáticas em C - Soma, subtração, multiplicação, divisão e módulo (ou resto da divisão) e precedência dos operadores
Operações matemáticas básicas. Fácil não? Por exemplo, quanto é: 1 + 1 x 2 ? Pode ser 3: 1 + (1x2) = 1 + 2 = 3 Ou pode ser 4: (1+1)x2 =...


O tipo char - escrevendo na linguagem C
Agora que você já sabe como lidar com inteiros e decimais na linguagem C, está na hora de estudarmos como escrever caracteres.

A função scanf - recebendo números do

```
 if(ptr == NULL)
  printf("Sem espaço suficiente\n");

 return 0;
}
```

A realloc() exige que você envie um ponteiro para ela por que pode ser que o novo bloco de memória que você pediu não esteja adjacente ao bloco que você indicou através do ponteiro, então o computador vai buscar outro bloco de endereço, daí o endereço do ponteiro muda.

Esse ponteiro que você passou, não necessariamente tinha que ter sido usado para aloca memória, pois se ele apontar para NULL, por exemplo, a realloc() vai funcionar exatamente como a malloc().

Exemplo de código: Como usar a função realloc() em C

Crie um programa que armazene dinamicamente números fornecidos pelo usuário.
O programa deve perguntar quantos números o usuário quer adicionar e receber tais números.
Não desperdice memória e tempo, use a função realloc() para realocar memória sempre que o usuário quiser inserir mais números.

Vamos usar 3 variáveis.
Um inteiro 'opcao' para armazenar a opção do usuário no menu, outro inteiro 'size' que vai armazenar o tamanho do array de números e o ponteiro para inteiro, que irá receber o endereço do array alocado.

Na função menu(), simplesmente colocamos as opções de Sair, Colocar mais números ou Exibir a lista de números.
Essa função retorna o inteiro da escolha do usuário.

Vamos usar o retorno dessa função menu() dentro de um teste condicional switch que irá tratar cada opção descrita pelo usuário.
Esse switch está dentro de um laço do while, que só termina quando o usuário digitar 0 (opcao=0).

A primeira função é realoca(), que vai receber dois argumentos: o ponteiro que irá guardar o endereço do bloco alocado e o tamanho do array de inteiros. Porém, não vamos enviar o inteiro 'size', e sim seu endereço de memória, pois queremos que este inteiro seja alterado dentro da função realoca().

Dentro desta função nós perguntamos quantos números o usuário quer adicionar à lista, e armazenamos essa informação na variável 'add'.
Agora vamos realocar nosso bloco de memória, que antes era do tamanho '*size' (lembre-se que passamos o endereço de memória do inteiro, então para pegar o valor armazenado nesse endereço usamos *size em vez de size, como ensinamos em nossas seção sobre Ponteiros em C de nossa apostila).

Depois vamos pedir esses números para os usuários, e vamos adicionar os números no array. Fazemos isso através da variável 'count' dentro de um laço for. O count vai de 0 até (add-1). Então esses novos

usuário
Até o momento, os artigos de nosso curso C Progressivo tem mostrado diversos programas, porém todos estáticos, sem controle e sem interação...

Lendo arquivos em C: As funções fgetc, fscanf e fgets
Agora que já aprendemos a escrever em arquivos em C , vamos aprender agora em nossa apostila de C a outra parte: aprender como ler informaç...


Programa: Criando uma calculadora em C
No artigo passado, sobre o laço DO WHILE em C , propomos para você um exercício: No artigo sobre o teste condicional SWITCH em C , mostra...

Buffer: o que é, como limpar e as funções fflush e __fpurge
No artigo passado foi pedido o seguinte programa: Fazer um programa em C que peça dois caracteres ao usuário e os exiba. Porém, há um pr...

números vão sempre no final do array, a partir da posição (*size) até a (*size + add - 1).

Feito isso, temos que mudar o valor da 'size', pois nosso array cresce. Era *size, agora é (*size + add), fazemos isso assim: *size += add
E feito, retornamos o novo ponteiro, com os novos números adicionados à lista.

Vale lembrar que após usar a realloc() devemos checar se seu computador conseguiu espaço em memória.
Se conseguiu, o teste if(ptr) retorna valor lógico TRUE, pois 'ptr' não é NULL, e fazemos a alocação.
Caso não tenha conseguido alocar o espaço de memória, o ponteiro 'ptr' vai apontar para NULL e o teste será falso, indo para o else que termina o programa.

Por fim, a função exib() recebe o ponteiro e o tamanho do array, e simplesmente exibe todos seus elementos.
Sem segredo.
E, como não podemos esquecer, liberamos a memória alocada, que é pontada pelo ponteiro 'ptr', através da função free().

Logo, o código de nosso programa em C fica:

```c
#include <stdio.h>
#include <stdlib.h>

int* realoca(int* ptr, int* size)
{
 int count,
  add;

 printf("Deseja adicionar quantos numeros: ");
 scanf("%d", &add);

 ptr = (int *) realloc(ptr, (*size + add)*sizeof(int) );
 if(ptr)
 {

  for(count=0 ; count < add ; count++)
  {
   printf("Numero [%d]: ", count+1);
   scanf("%d", &ptr[*size + count]);
  }

  *size += add;
 }
 else
 {
  printf("Espaço em memória insuficiente\n");
  free(ptr);
  exit(1);
```

```
 }
 return ptr;
}

void exibe(int* ptr, int size)
{
 int count;
 for(count=0 ; count<size ; count++)
  printf("%3d", ptr[count]);

}

int menu()
{
 int opcao;

 printf("\nO que deseja: \n");
 printf("0. Sair\n");
 printf("1. Adicionar numeros\n");
 printf("2. Exibir lista de numeros\n");
 printf("Opcao: ");
 scanf("%d", &opcao);

 return opcao;
}

int main(void)
{
 int opcao,
  size=0,
  *ptr=NULL;

 do
 {
  switch(menu())
  {
   case 0:
     opcao=0;
     break;

   case 1:
     ptr=realoca(ptr, &size);
     break;

   case 2:
     exibe(ptr, size);
     break;
```

```
    default:
      printf("Opcao invalida!\n");

   }
  }while(opcao);

  free(ptr);

  return 0;
}
```

A função calloc()

Para finalizar nossos estudos sobre alocação dinâmica de memória, vamos falar da função calloc() que é parecidíssima com a malloc(). Uma dessas diferenças é na sintaxe, porém seu propósito é o mesmo: alocar blocos de bytes em memória.

A sintaxe da calloc() é:
calloc(numero, tamanho_em_bytes);

Lembre-se que na malloc fazíamos: malloc(numero * tamanho_em_bytes)
Também usamos casting nos ponteiros.

Por exemplo, para alocar 'n' blocos de inteiros, com a calloc fazemos:
ptr = (int *) calloc(n, sizeof(int));

O mesmo para float:
ptr = (float *) calloc(n, sizeof(float));

Diferença entre calloc() e malloc()

Se pensar um pouco, a função calloc() é praticamente igual à malloc().
Porém, há uma pequena diferença.

Quando usamos a malloc() simplesmente reservamos um espaço de memória.
Já quando usamos a calloc(), além de reservar esse espaço de memória ele muda os valores contidos nesses bytes, colocando todos para 0. É como se usássemos a função memset(), que inicializa um bloco de memória com valor 0.

Então, quando devemos usar malloc() e calloc()?
Se você tiver que inicializar um bloco de memória com 0, faça isso usando a calloc(), pois é mais simples e otimizado que fazendo isso manualmente. Na verdade, enquanto seu computador está ocioso ele faz com que alguns espaços de memória recebam esse valor 0, e quando você for usar a calloc() ele vai procurar um bloco que esteja 'zerado', então, no geral, a calloc() é mais rápida e otimizada que a malloc().

A calloc() é muito usada para se trabalhar com vetores multidimensionais (matrizes), pois facilita para alocar uma certa quantidade de números de vetores, por exemplo.

Outro uso é no quesito segurança.
No tutorial passado de nosso curso nós mostramos que após usar um bloco de memória devemos apontar o ponteiro para NULL, pois senão ele continua apontando para o local antigo da memória, e isso seria uma grave falha, um brecha no sistema.
A vantagem da calloc() é que ele apaga os dados que existiam antes naquele bloco de memória, fazendo assim uma segurança maior, pois apaga as informações anteriores.

Exercícios propostos com calloc() e realloc()

1. Defina uma função chamada callocc(), que faz exatamente o que a calloc() faz.
Faça ela usando as funções malloc() e a memset(), que recebe três argumentos (o ponteiro, o que queremos colocar em todas as posições do vetor e o número de bytes):
memset(ptr, '\0', numero * tamanho_em_bytes);

2. Crie um programa que forneça os números de Fibonacci, o usuário escolhe o n-ésimo termo, e você fornece.
Calcule usando a realloc() para alocar memória.
Crie uma opção de modo que o usuário pode pedir outro termo, e esse novo termo deve ser achado da maneira mais eficiente possível (sem ter que recalcula todos os elementos de novo, pois os elementos da consulta anterior ainda estão no vetor - lembre-se que a realloc() aloca novos bytes e não apaga os anteriores).

O termo 't(n)' da sequência de Fibonacci é a soma dos dois anteriores
t(n) = t(n-1) + t(n-2), onde t(0)=0 e t(1)=1.

A sequência de Fibonacci revela coisas interessantíssimas sobre diversos fatos da natureza:
http://pt.wikipedia.org/wiki/N%C3%BAmero_de_Fibonacci

Tags: Alocação dinâmica de memória, calloc, Função, realloc

Nenhum comentário:

Postar um comentário

Postagem mais recente Página inicial Postagem mais antiga

Mas para nosso projeto se manter é preciso divulgação.
Para isso, basta curtir nossa página no Facebook e/ou clicar no botão +1 do Google.
Contamos e precisamos de seu apoio.

Publicidade

Melhor visualizado com Google **Chrome** e Mozilla **Firefox**